An opcode implementation of a finite difference viscothermal time-domain model of a tube resonator for wind instrument simulations

Alex Hofmann[1], Sebastian Schmutzhard[2], Montserrat Pàmies-Vilà[1], Gökberk Erdoğan[3], and Vasileios Chatziioannou[1] *

[1] Dept. of Music Acoustics, University of Music and Performing Arts Vienna, Austria
[2] Acoustics Research Institute, Austrian Academy of Sciences, Vienna, Austria
[3] Dept. of Electrical & Electronics Engineering Boğaziçi University, Istanbul, Turkey
corresponding author: hofmann-alex@mdw.ac.at

Abstract. This paper presents an opcode for Csound, that is based on a physical time-domain model of a closed-open tube resonator which is capable of simulating wind instruments like clarinets or saxophones. The tube model hereby considers sound radiation parameters as well as viscothermal losses that occur inside the tube. The model was implemented in C++ using the *Csound Plugin Opcode Framework*. The `resontube` opcode allows users to provide complex geometries for the model construction in k-time together with arguments for sound radiation and pick-up position. The opcode is published together with its source code as a git repository including documentation and examples.

Keywords: csound, opcode, physical model, resonator, tube

1 Introduction

Physical modelling-based sound synthesis is an estabilished technique in the field of computer music [11] and is well supported by a large number of opcodes in Csound [7]. A majority of physical models available for Csound are based on the digital waveguide method[4] (e.g. opcodes `wgclar, wgflute, wgbow` and opcodes from the Synthesis Toolkit (STK) by Cook and Scavone [4,9]).

Digital waveguides are computationally efficient algorithms that use simple delay lines to model a wave travelling in space. When the wave hits a boundary it is reflected. Depending on the boundary condition the sound wave is damped and may change its phase. In digital waveguides, boundary conditions are modelled by inserting filters into the delay line that mimic the effect of the respective boundary. The abstraction of the complex physical phenomena that happen with

* This research was supported by the Austrian Science Fund (FWF) P28655-N32 and the"mdw Call fuer Artistic Research Projekte" by the University of Music and Performing Arts Vienna. The authors like to thank Rory Walsh and Steven Yi for their support via the Csound Slack Chat.

[4] http://www.csounds.com/manual/html/SiggenWavguide.html

musical instruments as simple delay+filter algorithms allows for computationally efficient code that can easily run in real-time on consumer computers. However, this may restrict the model complexity and the quality of the sound [6].

The general approach to physical modelling directly considers the differential equations that describe the oscillating system. These can be solved numerically, using a variety of methods [12]. The prevalent approach in the last decades is to discretise the model equations using the finite difference method [2,1]. This approach might require a large number of computations but is capable of producing more realistic sounds. As of our knowledge, there are currently only three opcodes in Csound that make use of finite differences for physical modelling of musical instruments, namely `barmodel` (model of a metal bar), `prepiano` (model of a prepared piano string), and `platerev` (model of a resonating two dimensional rectangular plate).

This paper presents a new opcode to extend Csound by a mathematical time-domain wave propagation model for a closed-open tube resonator that takes viscothermal losses into account and allows for a varying cross-sectional area, as found in wind instruments such as clarinets or saxophones [10].

2 Tube Resonator Model

This section gives a short summary of the physical tube resonator model that is the basis for the new opcode. A detailed description, including a validation of this model can be found in [10].

A tube of length L is considered, with a cross-sectional area $S(x), 0 \leq x \leq L$. The time-domain model for the dynamics of the pressure p and the particle velocity v is given by

$$\partial_x p + \rho \partial_t v + z_v * v = 0, \qquad \partial_x (Sv) + \frac{S}{\rho c^2} \partial_t p + S y_\theta * p = 0, \tag{1}$$

where * denotes a convolution with respect to time, and the functions z_v and y_θ are the time domain versions of the series impedance Z and shunt addmitance Y. The boundary conditions are given by $v(t,0) = v_{\text{in}}(t)$ at $x = 0$ and for $x = L$

$$p(t,L) = S(L) z_r * v(t,L), \tag{2}$$

where z_r is a stipulated radiation impedance. The convolutions $z_v * v$ and $y_\theta * p$ are related to the viscothermal losses along the tube. For the computational algorithm used in this opcode, we replace equation (1) by the approximation

$$\partial_x p + \rho \partial_t v + R_0 v + \sum_{k=1}^{K} w_k = 0, \qquad \partial_x (Sv) + \frac{S}{\rho c^2} \partial_t p + S \sum_{k=1}^{K} q_k = 0, \tag{3a}$$

$$\text{where} \quad w_k(t) = R_k \int_0^t e^{-L_k (t-\tau)} \partial_t v(\tau) d\tau, \quad k = 1, \ldots, K \tag{3b}$$

$$\text{and} \quad q_k(t) = G_k \int_0^t e^{-C_k (t-\tau)} \partial_t p(\tau) d\tau, \quad k = 1, \ldots, K. \tag{3c}$$

Techniques for the computation of the coefficients R_k, L_k, G_k and C_k are discussed in [10,3]. We set $K = 4$, since we observed that taking K larger than four does not audibly change the result. The boundary condition is approximated by

$$S(L)R_r\partial_t v(t,L) = L_r p(t,L) + \partial_t p(t,L), \tag{4}$$

for certain coefficients R_r and L_r, see [2]. Using finite differences to approximate the derivatives, we compute approximations p_m^n, v_m^n, $w_{k,m}^n$ and $q_{k,m}^n$ to the solutions p, v w and q of (3), respectively, at discrete points (t_n, x_m) in time and space, where $t_n = n\Delta_t$, $n = 0, 1, 2\ldots$ and $x_m = m\Delta_x$ for $m = 0, \ldots, M$ and $L = M\Delta_x$, for fixed Δ_t and Δ_x. p_m^{n+1} and v_m^{n+1} are iteratively computed from results obtained at previous time steps. The derivation of the finite difference scheme is given in [10]. The boundary condition on the left gives for v_0^{n+1}

$$v_0^{n+1} = v_{\text{in}}^{n+1}. \tag{5}$$

Equation (3a) is discretised by finite differences. We compute $v_m^{n+1}, m = 1, \ldots, M$ from

$$\begin{aligned} &\frac{p_m^n - p_{m-1}^n}{\Delta_x} + \rho\frac{v_m^{n+1} - v_m^n}{\Delta_t} + R_{0,m}v_m^{n+1} + \\ &\sum_{k=1}^{K}\left[e^{-L_{k,m}\Delta_t}w_{k,m}^n + R_{k,m}(v_m^{n+1} - v_m^n)e^{-L_{k,m}\frac{\Delta_t}{2}}\right] = 0. \end{aligned} \tag{6}$$

and p_m^{n+1}, $m = 0, \ldots, M-1$ from

$$\begin{aligned} &\frac{S_{m+1}v_{m+1}^{n+1} - S_m v_m^{n+1}}{\Delta_x} + \frac{S_m}{\rho c^2}\frac{p_m^{n+1} - p_m^n}{\Delta_t} + \\ &S_m\sum_{k=1}^{K}\left[e^{-C_{k,m}\Delta_t}q_{k,m}^n + G_{k,m}(p_m^{n+1} - p_m^n)e^{-C_{k,m}\frac{\Delta_t}{2}}\right] = 0. \end{aligned} \tag{7}$$

Taking finite differences in (4) yields for $m = M$

$$S_M R_r\frac{v_M^{n+1} - v_M^n}{\Delta_t} = L_r p_M^{n+1} + \frac{p_M^{n+1} - p_M^n}{\Delta_t}, \tag{8}$$

from which p_M^{n+1} can be computed. Finally for $k = 1, \ldots, K$ and $m = 1, \ldots, M$, $w_{k,m}^{n+1}$ and $q_{k,m}^{n+1}$ are updated by

$$w_{k,m}^{n+1} = e^{-L_{k,m}\Delta_t}w_{k,m}^n + R_{k,m}(v_m^{n+1} - v_m^n)e^{-L_{k,m}\frac{\Delta_t}{2}}, \tag{9}$$

and

$$q_{k,m}^{n+1} = e^{-C_{k,m}\Delta_t}q_{k,m}^n + G_{k,m}(p_m^{n+1} - p_m^n)e^{-C_{k,m}\frac{\Delta_t}{2}}. \tag{10}$$

3 Opcode Implementation

The tube model is implemented in C++ following the *Csound Plugin Opcode Framework* [6]. All code is made available as a public git repository [5]

The implementation of the `resontube` opcode, a tube resonator with viscothermal losses as described in Section 2, can be found in the file `resonators/resontube.cpp`. A library of functions is given in `src/tube.cpp` and constants are in `src/const.cpp`. Following, we give an overview of the implementation of the model as an opcode for Csound.

Initialisation: When the opcode is loaded (`init()`), an equispaced grid in the longitudinal direction of the tube with M grid points is instantiated. Based on the user given geometry, the cross sectional area S is calculated for each grid point (see Figure 1). The grid consists of five csound arrays (`csnd::AuxMem<MYFLT>`) in which the status of the tube is processed. The size of the arrays is initially allocated for Mmax=400 (`src/const.cpp`) so that no additional memory needs to be allocated during runtime, even when the user is changing the length of the tube. The five main arrays are `vnew` for the particle velocity ($v_m^{n+1}, m = 1, \ldots, M$ from eq.6), `pnew` for the air pressure (p_m^{n+1} from eq. 7), `S` for cross sectional area at each grid point (S_m), and `qloss` and `wloss` for the viscothermal loss related variables ($w_{k,m}^{n+1}$ and $q_{k,m}^{n+1}$ given in eq. 9 & 10). `AuxMem` iterators (e.g. `csnd::AuxMem<MYFLT>::iterator iter_pnew`) are created for each array. Computations on the arrays are only done within the range $m = 0, \ldots, M$. As a two-point scheme in time is used, the algorithm requires a memory of the previous tube state. Therefore, a copy of all grid values needs to be preserved in additional arrays (`pold, vold, qlossold, wlossold`) with the respective iterators.

In this model, two types of losses are calculated. A) radiation losses (`rad_alphaS`) at the end of the tube where the sound is radiated out of the tube, depending on the cross sectional area at the last grid point as given in eq.(4). B) viscothermal losses that apply at each grid point along the virtual tube are calculated. Factors for the convolutions in eq.(1) are prepared in the function `compute_loss_arrays()`.

Runtime: During runtime (`aperf()`), it is checked if any of the k-rate input arguments (length, geometry, see Section 4 for details) were changed by the user. If this happens, the grid has to be re-computed. This involves calculating the required number of grid points (M < Mmax), the spacing of the grid points (dx), the cross sectional area (S) at each grid point (x), as well as the losses. To maintain the wave inside the tube (pressure, velocity, losses), all new grid arrays are updated with interpolated values taken from the preserved grid arrays.

[5] `https://github.com/ketchupok/half-physler/tree/visco_pointers`. Currently, the repository holds three slightly different opcodes. Two are tube resonators without viscothermal losses, used in a different project [5], where `halfphysler_bela` is specifically optimised to run on the ultra-low latency embedded computing platform Bela [8].

In the audio-loop (`for (auto & o_sound : out_sound) {...}`), the wave propagation in the tube is computed for each time step (sample), and the audio I/Os are assigned. An input signal is given to the model as a particle velocity to the closed end of the tube (`vnew[0] = in;`). Two different audio outputs are assigned. `Out_Feedback` returns the pressure at the beginning of the tube (`pnew[0]`) prior to computing the next time step, whereas `o_sound` is returning the pressure at a variable grid point `pnew[x]`, $x < M$ after the update function `update_visco()`[6]. The function `update_visco()` updates the pressure and velocity properties according to equations (6) and (7). `Update_losses()` is updating the loss arrays following equations (9) and (10). Finally the status of the grid is copied to `pold & vold` to be preserved for the next call of `aperf()`.

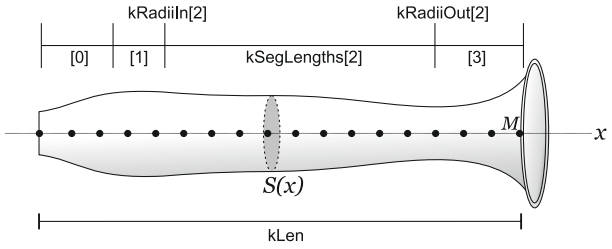

Fig. 1. Schematic of the tube model with a closed end at the left side and an open end at the right side. Geometry is given in segments as Csound arrays. Here an example with four segments from which the cross-sectional area (S) is computed for each grid point.

4 Usage

The `resontube` opcode implements a tube resonator with one closed and one open end, similar to the resonator of a clarinet or a saxophone. An overview of all user parameters of the opcode including a description of the underlying physical parameters is given in Table 1. In Csound the opcode is called by:

```
aFeedb, aSnd resontube aVelocity, kLen, kSegLengths[], kRadiiIn[],
                       kRadiiOut[], kCurveType[], [kEndReflect,
                       kDensity, kPickPos, kComputeVisco]
```

The resonator is driven by an input particle velocity (`aVelocity`), a parameter that describes the speed of the air entering the tube, for example via a single-reed instrument mouthpiece. The second input parameter `kLen` was introduced, to allow to change the resonance frequency of the resonator in a woodwind

[6] To save CPU, computation of all viscothermal losses can be turned off (`computeVisco=0`), and respectively the function `update_vp()` is called

instrument-like style. A given `kLen` in meters cuts the resonator at this point, similar to the function of opening toneholes at acoustic woodwind instruments.

The initial geometry of the entire resonator is given in segments via the Csound arrays `kSegLengths[], kRadiiIn[], kRadiiOut[], kCurveType[]`. Each segment is defined by its length, input radius, output radius and an interpolation curve type. We allow up to 25 segments. The example below gives a good approximation of a Bb-flat clarinet geometry using only 4 segments.

```
kSegLengths[] fillarray 0.0316, 0.051, .3, 0.02
kRadiiIn[] fillarray 0.0055, 0.00635, 0.0075, 0.0075
kRadiiOut[]  fillarray 0.0055, 0.0075, 0.0075, 0.0275
kCurveType[] fillarray 1, 1, 1, 2
```

Additional parameters to shape the sound by modifying the end reflection, the air density and the pick-up position along the tube are provided as k-rate inputs. An option to switch between the computation of viscothermal losses or not was added, which allows real-time playback also for complex geometries and long resonators on consumer PCs or embedded platforms like *Bela* [8].

5 Discussion

The presented opcode extends Csound by a finite difference model of a tube resonator, similar to resonators we find in clarinets or saxophones. Publishing the model for Csound allows live-electronic performers, composers or instrument makers to explore numerical modelling in an environment that handles I/O management and allows to combine physical modelling with other signal processing opcodes. Exciting experiments are possible when using Csound's capability of creating an internal feedback (ksmps=1), as shown in one of the online examples. A future version of the opcode could involve an extension to an open-open tube (flute) resonator, as well as adding details like tonehole geometry or register key modeling.

References

1. S. Bilbao. Direct simulation of reed wind instruments. *Computer Music Journal*, 33(4):43–55, 2009.
2. S. Bilbao. *Numerical sound synthesis.* John Wiley & Sons, 2009.
3. S. Bilbao and R. Harrison. Passive time-domain numerical models of viscothermal wave propagation in acoustic tubes of variable cross section. *JASA*, 140(1):728–740, 2016.
4. P. Cook. *Real Sound Synthesis for Interactive Applications.* AK Peters, 2002.
5. A. Hofmann, V. Chatziioannou, S. Schmutzhard, G. Erdoğan, and A. Mayer. The half-physler. In *Proc. NIME 2019*, page (accepted), Porto Allegre, BR, 2019.
6. V. Lazzarini. The csound plugin opcode framework. In *SMC*, pages 267–274, 2017.
7. V. Lazzarini, S. Yi, J. ffitch, J. Heintz, Ø. Brandtsegg, and I. McCurdy. *Physical Models*, pages 385–405. Springer International Publishing, Cham, 2016.

8. A. McPherson, R. Jack, and G. Moro. Action-sound latency: Are our tools fast enough? In *Proc. NIME*, volume 16, pages 20–25, Brisbane, Australia, 2016.
9. G. P. Scavone and J. O. Smith. A stable acoustic impedance model of the clarinet using digital waveguides. In *Proc. DAFx–06*, pages 89–94, 2006.
10. S. Schmutzhard, V. Chatziioannou, and A. Hofmann. Parameter optimisation of a viscothermal time-domain model for wind instruments. In *Proc. ISMA*, pages 27–30, 2017.
11. J. Smith. Viewpoints on the history of digital synthesis. In *Proc. ICMC*, pages 1–10, 1991.
12. V. Välimäki, J. Pakarinen, C. Erkut, and M. Karjalainen. Discrete-time modelling of musical instruments. *Reports on Progress in Physics*, 69(1):1–78, 2006.

Table 1. User parameters for the closed-open tube model.

Variable	type	Parameter	Range	Functionality
aVelocity	a-rate	v		Input particle velocity (m/s)
kLen	k-rate	L	0.01–3 m (with sr = 44100)	length of the tube
kSegLengths[]	k-rate	L_{part}	$\sum =<$ 3 m	length of each segment, given as Csound array
kRadiiIn[]	k-rate	$r_{\text{part-in}}$	0.0075–0.0095 m (= 7.5–9.5 mm)	radius at the beginning of each segment, given as Csound array
kRadiiOut[]	k-rate	$r_{\text{part-out}}$	0.0075–0.0095 m (= 7.5–9.5 mm)	end-radius of segment, given as Csound array
kCurveType[]	k-rate		1)=linear, 2)=parabolic, 3)=exponential	interpolation mode for computation of S, given as Csound array
kEndReflection	k-rate	α	0.1–4.0	multiplier for end reflection coefficient (radiation resistance)
kDensity	k-rate	β	0.1–30.0	multiplier for air density
kPickPos	k-rate	m	0.0–1.0	scaled pickup position along the tube
kComputeVisco	k-rate		0 / 1	turn On/Off the computation of viscothermal losses